

Instagram

Michael Chiang - 861126014

Tinh La - 861199635

Mingi Lee - 862019083

CS 179g Project in Computer Science - Database Systems 001/021

Professor Petko Bakalov

TA Runze Li

December 18, 2020

Table of Contents

Project Summary.....	3
Purpose.....	3
Requirements.....	3
Specifications.....	4
Extras Functionality.....	17
ER Diagram.....	17
Justifications.....	18
Entities.....	18
Relations.....	19
SQL vs NoSQL.....	20
Schema.....	20
Justifications.....	25
Implementation.....	25
Postgresql.....	25
MongoDB.....	26
Python.....	26
psycopg2.....	27
pymongo.....	28
gridfs.....	28
Sample Data.....	29
User Manual.....	30

Project Summary:**Purpose:**

This project is a photo-sharing service much like Instagram, where users can upload photos and share them with other users. We are building this project to interact with and learn how to handle a hybrid SQL and NOSQL environment. To accomplish this we have implemented Postgresql and MongoDB into our design to handle their respective sql natures.

Instagram is a social networking service which enables its users to upload and share their photos and videos with other users. Instagram users can choose to share information either publicly or privately. Anything shared publicly can be seen by any other user, whereas privately shared content can only be accessed by a specified set of people. Instagram also enables its users to share through many other social networking platforms, such as Facebook, Twitter, Flickr, and Tumblr.

For the sake of this exercise, we plan to design a simpler version of Instagram, where a user can share photos and can also follow other users. The 'News Feed' for each user will consist of top photos of all the people the user follows.

Functional Requirements:

1. Users should be able to upload/download/view photos.
2. Users can perform searches for other users based on photo titles, tags, ratings and so on.
3. Users can follow other users.
4. The system should be able to generate and display a user's News Feed consisting of top photos from all the people the user follows.
5. The system should allow adding tags to photos.
6. The system should allow searching photos on tags, ratings, dates or publishing users.

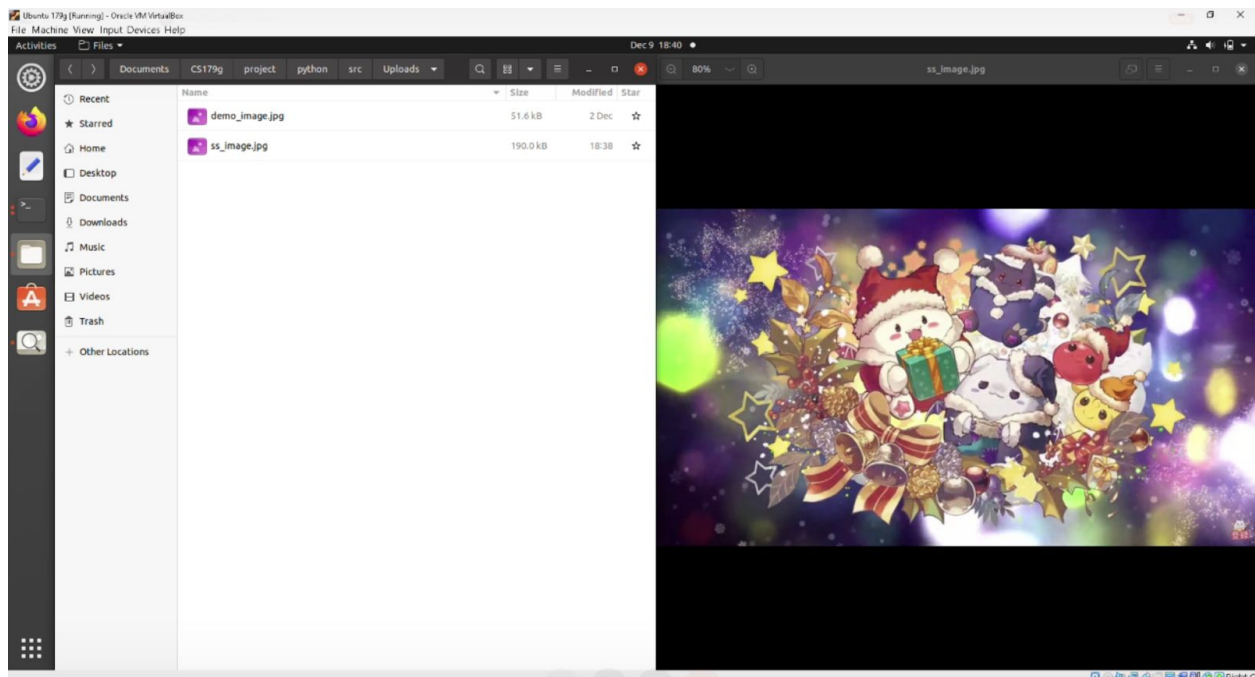
7. The system should allow commenting on photos.
8. The system should allow tagging users to photos.
9. Our services should be able to record stats of photos, e.g., likes/dislikes, total number of views, etc.
10. System should be able to list the most popular photos and users.

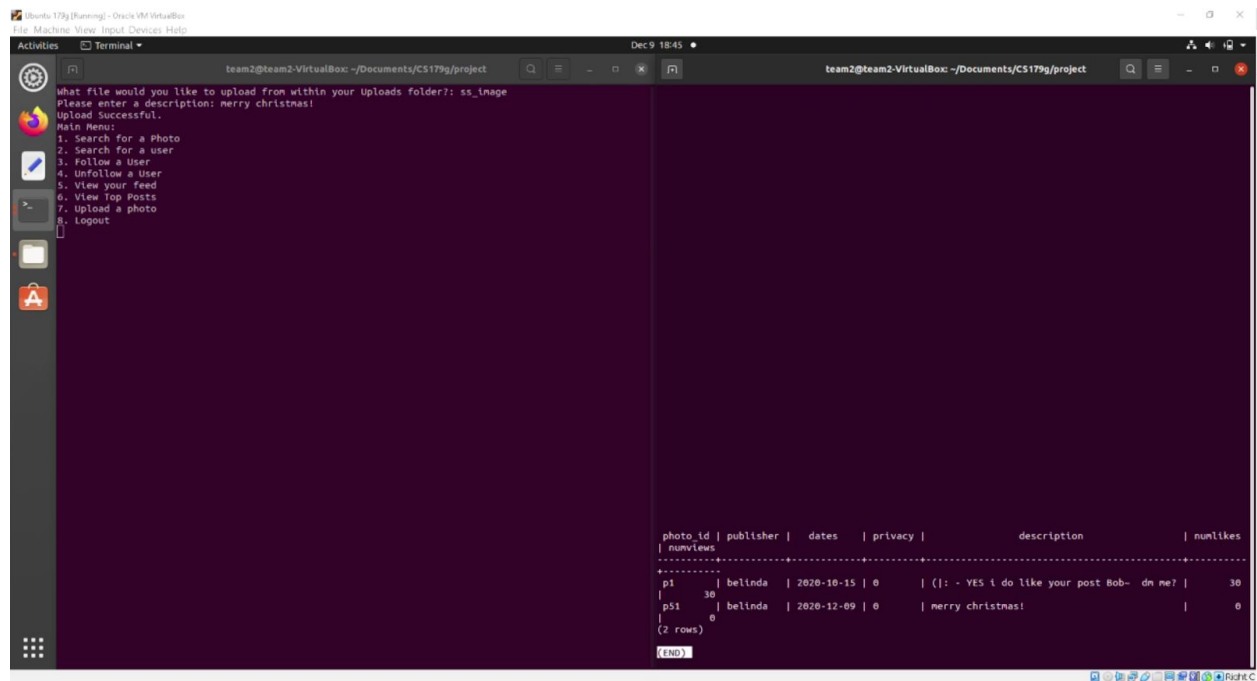
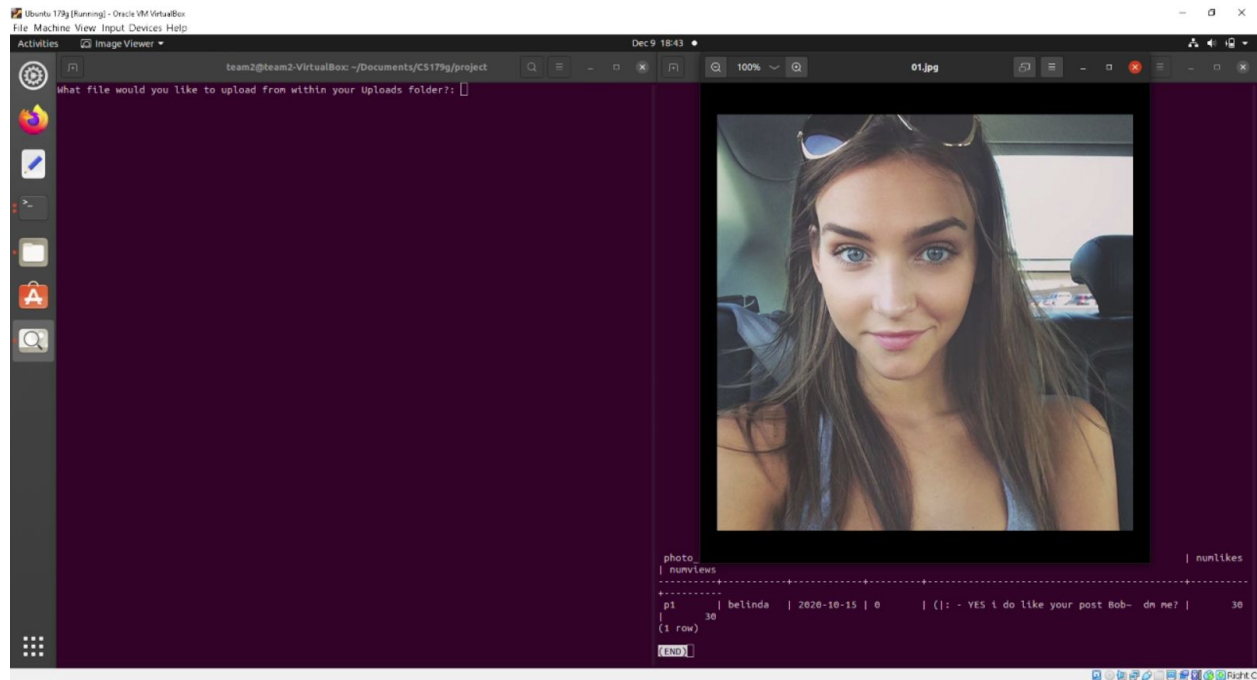
Non-functional Requirements:

1. Our service needs to be highly available.
2. Consistency can take a hit (in the interest of availability), if a user doesn't see a photo for a while; it should be fine.
3. The system should be highly reliable; any uploaded photo or video should never be lost.

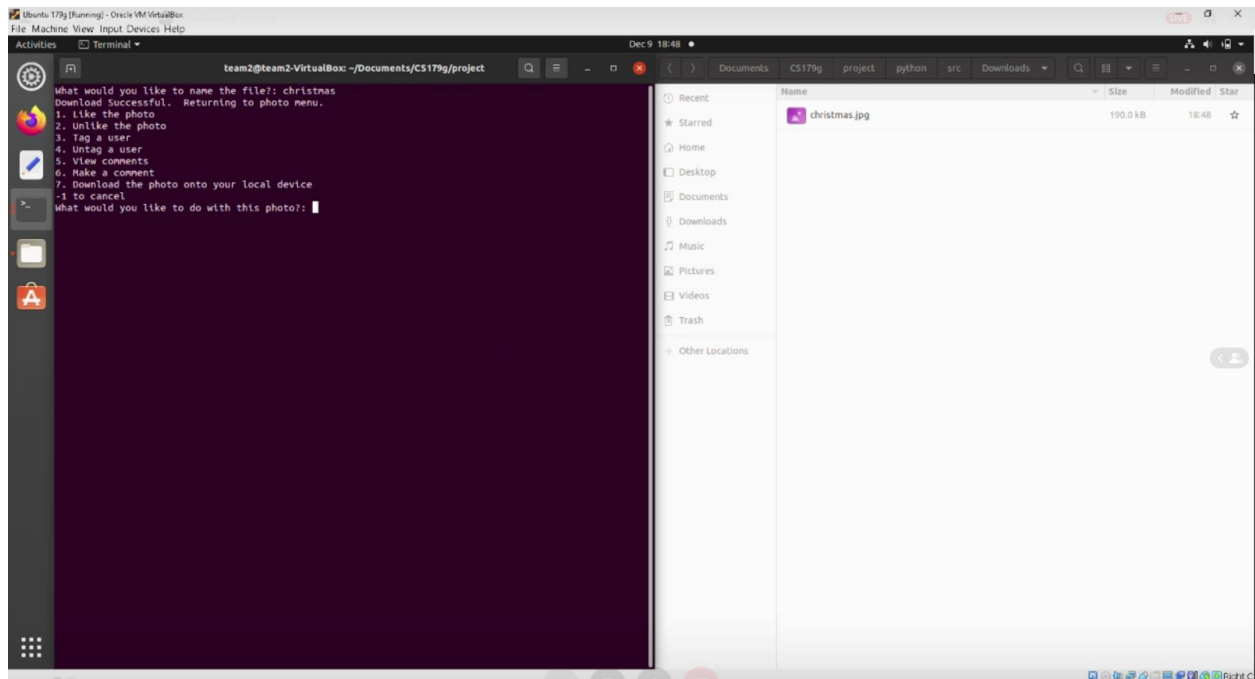
Specifications: (What can our system do)

- Users are able to upload photos, this is handled by upload.py.
 - Fulfils Functionality Requirement #1





- Users are able to download photos, this is handled by download.py
 - Fulfills Functionality Requirement #1



- Users are able to view photos, this handled by searchPhotoMenu.py
 - After searching, the user is given a list of photos they can choose from to view.
 - Fulfills Functionality Requirement #1



- Users are able to search for other users based on their username, the photo id, or the description of their photos.
 - Fulfills Functionality Requirement #2

```

team2@team2-VirtualBox: ~/Documents/CS179g/project
Enter the number of followers to search by: 15
username: belinda
username: bob
Do you want to search for another user? [Y/N]:

team2@team2-VirtualBox: ~/Documents/CS179g/project
psql -h localhost impostergram_db
Type "help" for help.

impostergram_db=# SELECT * FROM Users WHERE username = 'belinda';
-----
username | email          | pass | numfollows
-----
belinda  | belinda@gmail.com | 111  | 28
(1 row)

impostergram_db=# SELECT * FROM Photos WHERE publisher = 'belinda';
impostergram_db=# SELECT * FROM Users WHERE publisher = 'belinda';
impostergram_db=# SELECT * FROM Users;
-----
username | email          | pass | numfollows
-----
belinda  | belinda@gmail.com | 111  | 28
bob      | bob@gmail.com   | 121  | 29
kyle     | kyle@gmail.com  | 222  | 0
cindy    | cindy@gmail.com | 131  | 1
chris    | chris@gmail.com | 141  | 1
steven   | steven@gmail.com | 151  | 2
frank    | frank@gmail.com | 161  | 1
gellert  | gellert@gmail.com | 171  | 1
henry    | henry@gmail.com | 181  | 1
lilly    | lilly@gmail.com | 191  | 1
jacob    | jacob@gmail.com | 211  | 1
rory     | rory@gmail.com  | 212  | 1
lary     | lary@gmail.com  | 213  | 1
nary     | nary@gmail.com  | 214  | 1
nancy    | nancy@gmail.com | 215  | 1
jessie   | jessie@gmail.com | 216  | 1
steph    | steph@gmail.com | 217  | 1
calvin   | calvin@gmail.com | 218  | 1
nario    | nario@gmail.com | 219  | 1
peter    | peter@gmail.com | 311  | 1
raoul    | raoul@gmail.com | 312  | 1
brian    | brian@gmail.com | 313  | 1
conner   | conner@gmail.com | 314  | 1
mustafa  | mustafa@gmail.com | 315  | 1
tohsaka  | tohsaka@gmail.com | 316  | 3
mychul   | mychul@gmail.com | 317  | 2
billy    | billy@gmail.com | 318  | 1
ton      | ton@gmail.com   | 319  | 1
sally    | sally@gmail.com | 333  | 1
sharon   | sharon@gmail.com | 123  | 1
(30 rows)

impostergram_db=#
  
```

```

1. Search by username
2. Search by Photo ID
3. Search by Description
4. Search by greater than or equal to number of followers
5. Search by greater than or equal to number of likes
6. Search by the number of photos published
Which option do you want to search? (-1. To Exit):
  
```

- Users are able to (un)follow other users, this is handled by follows.py
 - Fulfills Functionality Requirements #3

Ubuntu 17g [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

team2@team2-VirtualBox: ~/Documents/CS179g/project

Please enter a username to Follow: raoul

steven	belinda
frank	belinda
gellert	belinda
henry	belinda
lilly	belinda
jacob	belinda
rory	belinda
lary	belinda
nary	belinda
nancy	belinda
jessie	belinda
steph	belinda
calvin	belinda
nario	belinda
peter	belinda
raoul	belinda
brian	belinda
conner	belinda
mustafa	belinda
tohsaka	belinda
mychul	belinda
billy	belinda
ton	belinda
kyle	belinda
sally	belinda
sharon	belinda
belinda	bob
cindy	bob
chris	bob
steven	bob
frank	bob
gellert	bob
henry	bob
lilly	bob
jacob	bob
rory	bob
lary	bob
nary	bob
nancy	bob
jessie	bob
steph	bob
calvin	bob
nario	bob
peter	bob
raoul	bob
brian	bob
conner	bob
mustafa	bob

Ubuntu 17g [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

team2@team2-VirtualBox: ~/Documents/CS179g/project

Please enter a username to Follow: raoul

Success!

Main Menu:

1. Search for a Photo
2. Search for a user
3. Follow a User
4. Unfollow a User
5. View your feed
6. View Top Posts
7. Upload a photo
8. Logout

calvin	bob
nario	bob
peter	bob
raoul	bob
brian	bob
conner	bob
mustafa	bob
tohsaka	bob
mychul	bob
billy	bob
ton	bob
kyle	bob
sally	bob
sharon	bob
lary	steven
jessie	tohsaka
jacob	tohsaka
mustafa	mychul
kyle	cindy
kyle	chris
kyle	steven
kyle	frank
kyle	gellert
kyle	henry
kyle	lilly
kyle	jacob
kyle	rory
kyle	lary
kyle	mary
kyle	nancy
kyle	jessie
kyle	steph
kyle	calvin
kyle	nario
kyle	peter
kyle	raoul
kyle	brian
kyle	conner
kyle	mustafa
kyle	tohsaka
kyle	mychul
kyle	billy
kyle	ton
kyle	sally
kyle	sharon
belinda	raoul

(89 rows)

- Users are able to see a feed of the top photo from each user they follow, this is handled by feeds.py
 - Currently the display for feed is uncapped; however, if this were to be built with a gui component we would only display a smaller section at a time and have the user cycle between pages. The reason for this is because currently our implementation to show pictures to the user can only handle one image at a time and would not be able to display multiple pictures like a feed. Instead we currently display information about the picture and the photo id. With this information the user can continue to searchPhotoMenu and view their specific pictures. Because of this limitation the uncapped nature of feed will not hit a big resource requirement even if the user follows a huge amount of other users as the retrieval and display is merely limited to text meta data.
 - Users can specify display their feed by # of photo views or by # of photo likes for each of the users they follow.
 - Fulfils Functionality Requirements #4

```
sort your feed by:
1. Views
2. Likes
-1. Return to main menu
```

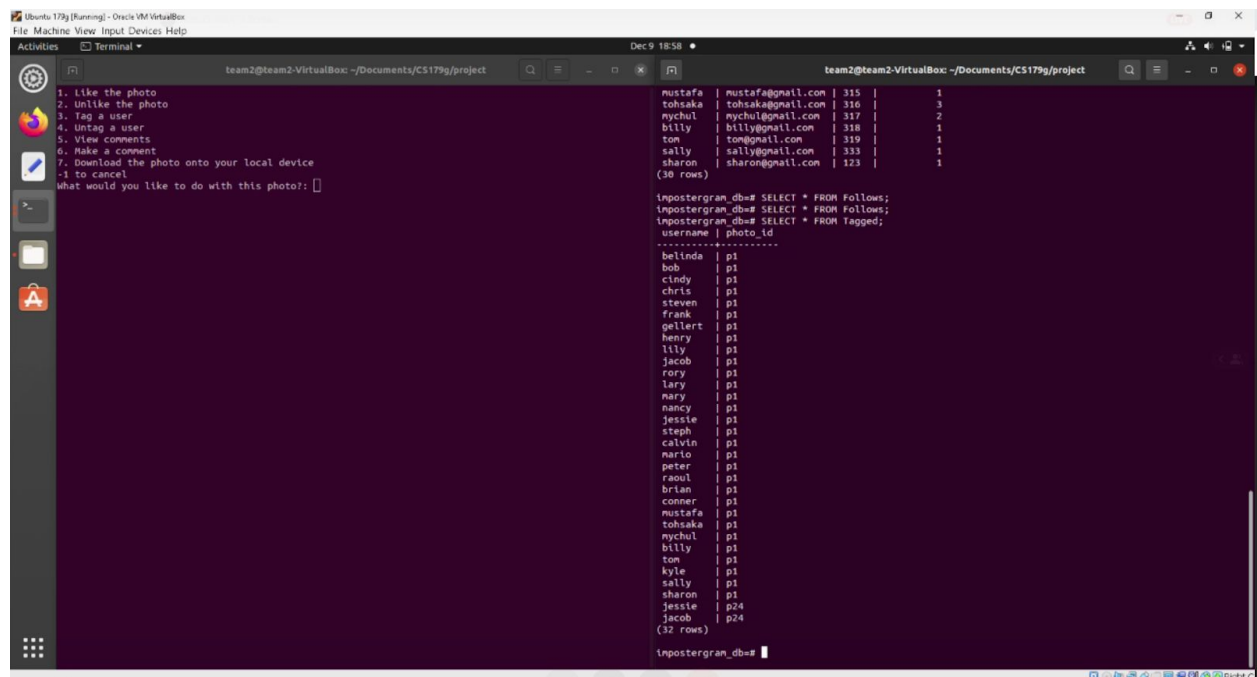
```
Sort your feed by:
1. Views
2. Likes
-1. Return to main menu
1
User belinda's most popular photo is p1
User bob's most popular photo is p2
User cindy's most popular photo is p3
User chris's most popular photo is p4
User steven's most popular photo is p5
User frank's most popular photo is p6
User gellert's most popular photo is p7
User henry's most popular photo is p8
User lily's most popular photo is p9
User jacob's most popular photo is p10
User rory's most popular photo is p11
User lary's most popular photo is p12
User mary's most popular photo is p13
User nancy's most popular photo is p14
User jessie's most popular photo is p15
User steph's most popular photo is p16
User calvin's most popular photo is p17
User mario's most popular photo is p18
User peter's most popular photo is p19
User raoul's most popular photo is p20
User brian's most popular photo is p21
User conner's most popular photo is p22
User mustafa's most popular photo is p23
User tohsaka's most popular photo is p24
User mychul's most popular photo is p25
User billy's most popular photo is p26
User tom's most popular photo is p27
User sally's most popular photo is p31
User sharon's most popular photo is p50
Sort your feed by:
1. Views
2. Likes
-1. Return to main menu
█
```

```
1. Views
2. Likes
-1. Return to main menu
2
User belinda's best photo is p1
User bob's best photo is p2
User cindy's best photo is p3
User chris's best photo is p4
User steven's best photo is p5
User frank's best photo is p6
User gellert's best photo is p7
User henry's best photo is p8
User lily's best photo is p9
User jacob's best photo is p10
User rory's best photo is p11
User lary's best photo is p12
User mary's best photo is p13
User nancy's best photo is p14
User jessie's best photo is p15
User steph's best photo is p16
User calvin's best photo is p17
User mario's best photo is p18
User peter's best photo is p19
User raoul's best photo is p20
User brian's best photo is p21
User conner's best photo is p22
User mustafa's best photo is p23
User tohsaka's best photo is p24
User mychul's best photo is p25
User billy's best photo is p26
User tom's best photo is p27
User sally's best photo is p31
User sharon's best photo is p50
Sort your feed by:
1. Views
2. Likes
-1. Return to main menu
```

Team 2: CS 179g Instagram

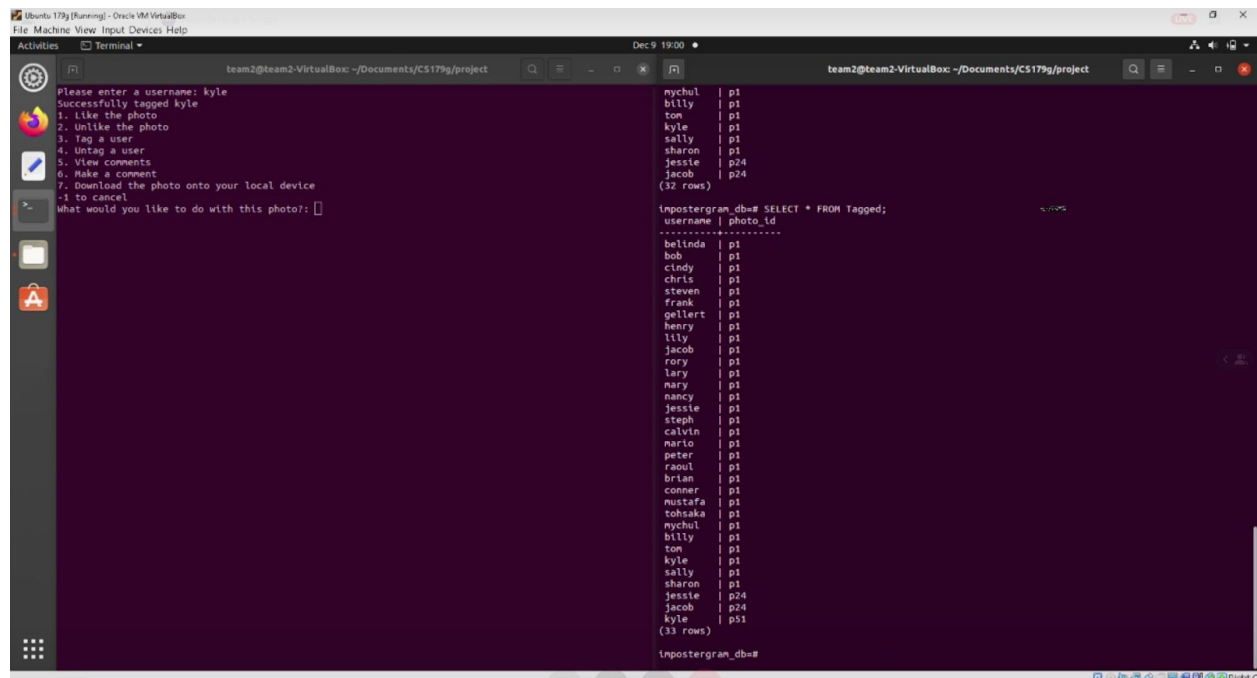
12

- Users are able to tag users to photos, this is handled in tagged.py
 - Fulfills Functionality Requirements # 5



```
1. Like the photo
2. Unlike the photo
3. Tag a user
4. Untag a user
5. View comments
6. Make a comment
7. Download the photo onto your local device
-1 to cancel
What would you like to do with this photo?:
```

username	photo_id
belinda	p1
bob	p1
cindy	p1
chris	p1
steven	p1
frank	p1
gellert	p1
henry	p1
lily	p1
Jacob	p1
rory	p1
Lary	p1
nary	p1
nancy	p1
Jessie	p1
steph	p1
calvin	p1
marlo	p1
peter	p1
raoul	p1
brian	p1
conner	p1
mustafa	p1
tohsaka	p1
mychul	p1
billy	p1
tom	p1
kyle	p1
sally	p1
sharon	p1
Jessie	p24
Jacob	p24



```
Please enter a username: kyle
Successfully tagged kyle
1. Like the photo
2. Unlike the photo
3. Tag a user
4. Untag a user
5. View comments
6. Make a comment
7. Download the photo onto your local device
-1 to cancel
What would you like to do with this photo?:
```

username	photo_id
belinda	p1
bob	p1
cindy	p1
chris	p1
steven	p1
frank	p1
gellert	p1
henry	p1
lily	p1
Jacob	p1
rory	p1
Lary	p1
nary	p1
nancy	p1
Jessie	p1
steph	p1
calvin	p1
marlo	p1
peter	p1
raoul	p1
brian	p1
conner	p1
mustafa	p1
tohsaka	p1
mychul	p1
billy	p1
tom	p1
kyle	p1
sally	p1
sharon	p1
Jessie	p24
Jacob	p24
kyle	p31

- ```
1. Search by a tagged username
2. Search by greater than # of likes
3. Search by less than # of likes
4. Search by a specific # of likes
5. Search by exact date
6. Search by range of dates
7. Search by username
8. View a specific photo
Which option do you want to search by? (-1. To Exit):
```

The screenshot shows a Kali Linux terminal window with the following content:

```

team2@team2-VirtualBox: ~/Documents/CS179g/project
Please enter a tagged user to search: kyle
user kyle tagged in photo id p1
user kyle tagged in photo id p51
which photo id would you like to see? (-1 to cancel):

mysql> SELECT * FROM Tagged;
+-----+-----+
| username | photo_id |
+-----+-----+
belinda	p1
bob	p1
cindy	p1
chris	p1
steven	p1
frank	p1
gellert	p1
henry	p1
lily	p1
jacob	p1
rory	p1
lary	p1
nary	p1
nancy	p1
jessie	p1
steph	p1
calvin	p1
marto	p1
peter	p1
raoul	p1
brian	p1
conner	p1
mustafa	p1
tohsaka	p1
nychul	p1
billy	p1
ton	p1
kyle	p1
sally	p1
sharon	p1
jessie	p24
jacob	p24
kyle	p51
+-----+-----+
(33 rows)

mysql>

```



- Users are able to view comments, create comments, and like comments, this is handled by comments.py, commentLikes.py and viewComments.py
  - Fulfills Functionality Requirement # 7

```

Please enter your comment: happy holidays!
Confirm comment: <happy holidays!-> (Y/N): y
Successfully commented.
1. Like the photo
2. Unlike the photo
3. Tag a user
4. Untag a user
5. View comments
6. Make a comment
7. Download the photo onto your local device
-1 to cancel
What would you like to do with this photo?:

sharon | p1
jessie | p24
jacob | p24
(32 rows)

inpostergram_db=# SELECT * FROM Tagged;
username | photo_id

belinda | p1
bob | p1
cindy | p1
chris | p1
steven | p1
frank | p1
gellert | p1
henry | p1
lily | p1
jacob | p1
rory | p1
lary | p1
nary | p1
nancy | p1
jessie | p1
steph | p1
calvin | p1
mario | p1
peter | p1
raoul | p1
brian | p1
conner | p1
mustafa | p1
tohsaka | p1
mychul | p1
billy | p1
tom | p1
kyle | p1
sally | p1
sharon | p1
jessie | p24
jacob | p24
kyle | p51
(33 rows)

inpostergram_db=# SELECT * FROM Comments WHERE photo_id = 'p51';
comment_id | comments | username | photo_id | dates | numlikes

(0 rows)

inpostergram_db=#

```

```

Please enter your comment: happy holidays!
Confirm comment: <happy holidays!-> (Y/N): y
Successfully commented.
1. Like the photo
2. Unlike the photo
3. Tag a user
4. Untag a user
5. View comments
6. Make a comment
7. Download the photo onto your local device
-1 to cancel
What would you like to do with this photo?:

username | photo_id

belinda | p1
bob | p1
cindy | p1
chris | p1
steven | p1
frank | p1
gellert | p1
henry | p1
lily | p1
jacob | p1
rory | p1
lary | p1
nary | p1
nancy | p1
jessie | p1
steph | p1
calvin | p1
mario | p1
peter | p1
raoul | p1
brian | p1
conner | p1
mustafa | p1
tohsaka | p1
mychul | p1
billy | p1
tom | p1
kyle | p1
sally | p1
sharon | p1
jessie | p24
jacob | p24
kyle | p51
(33 rows)

inpostergram_db=# SELECT * FROM Comments WHERE photo_id = 'p51';
comment_id | comments | username | photo_id | dates | numlikes

(0 rows)

inpostergram_db=# SELECT * FROM Comments WHERE photo_id = 'p51';
comment_id | comments | username | photo_id | dates | numlikes

334 | happy holidays!- | kyle | p51 | 2020-12-09 | 0
(1 row)

inpostergram_db=#

```

- Users are able to tag other users onto photos in our tagged.py which is called by our searchPhotoMenu.py. This is showcased above for functionality #5
  - Fulfills Functionality Requirement #8
- Our system keeps track of metadata related to photos and users. We have dedicated tables, tagged, views, photolikes, likes, and more to store this information on our postgres side of things.
  - Fulfills Functionality Requirement #9
- Users are able to see either a feed related to their followed users or see the top posts globally on the site. This is accomplished by feed.py and top.py using the data we mentioned above to fulfill requirement #9.
  - Fulfills Functionality Requirement #10

```
What would you like to view?
1. Top 5 Users
2. Top 5 Photos by Views
3. Top 5 Photos by likes
-1. Return to main menu
█
```

```
What would you like to view?
1. Top 5 Users
2. Top 5 Photos by Views
3. Top 5 Photos by likes
-1. Return to main menu
1
No.1 username: bob
Number of followers: 29

No.2 username: belinda
Number of followers: 28

No.3 username: tohsaka
Number of followers: 3

No.4 username: steven
Number of followers: 2

No.5 username: mychul
Number of followers: 2

What would you like to view?
1. Top 5 Users
2. Top 5 Photos by Views
3. Top 5 Photos by likes
-1. Return to main menu
2
No.1 photo id: p1
Number of views: 30

No.2 photo id: p2
Number of views: 29

No.3 photo id: p24
Number of views: 8

No.4 photo id: p5
Number of views: 5

No.5 photo id: p18
Number of views: 5

What would you like to view?
1. Top 5 Users
2. Top 5 Photos by Views
3. Top 5 Photos by likes
-1. Return to main menu
```

```
What would you like to view?
1. Top 5 Users
2. Top 5 Photos by Views
3. Top 5 Photos by likes
-1. Return to main menu
3
No.1 photo id: p1
Number of likes: 30

No.2 photo id: p24
Number of likes: 8

No.3 photo id: p18
Number of likes: 5

No.4 photo id: p5
Number of likes: 4

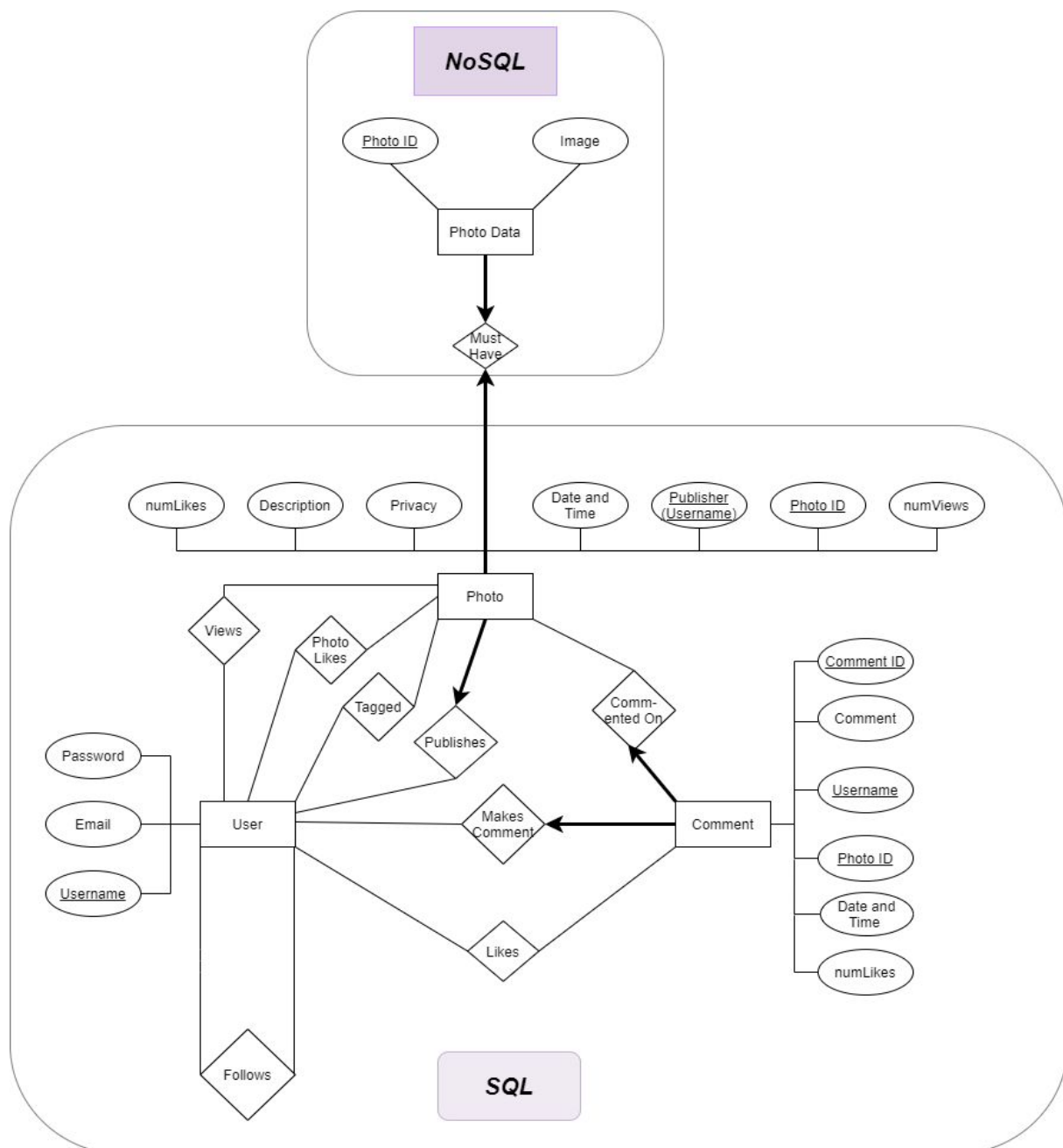
No.5 photo id: p28
Number of likes: 2

What would you like to view?
1. Top 5 Users
2. Top 5 Photos by Views
3. Top 5 Photos by likes
-1. Return to main menu
```



**Extra Functionality:**

- Users are able to login, create a new account through login.py
- Users are able to delete their own photos if they wish to do so through delete.py
- We originally planned to include a privacy option for users on their posted photos and this can be seen in our ER diagram, data and schema; however, we decided to forgo implementing this feature due to time constraints.

**ER Diagram:**

**Justifications:**

We are assuming that the service being supplied mimics the services that Instagram offers only to a lesser extent. In this case there are contradictions from what the requirements specify and the intended purpose of mimicking the services provided by Instagram. For now we will proceed with our design process matching closer to what Instagram offers and may return to update our own specs to meet the requirements if need be. This is because we believe that the intent of this is to create a copy or a lesser copy rather than a different photo-sharing service.

**Entities:**

User Entity: A user shall be composed of the following attributes: a username, a password, and a registered email. The primary key of this entity will be the username. Originally we debated between either the email or the username; however, we decided on username since Instagram does not allow duplicate usernames and since a user may wish to keep their email private having the primary key be the username allows for searching features to respect this potential wish for privacy. We also have a numFollows attribute that is update when the followers table is altered.

Photo Entity: A photo shall be composed of the following attributes: a photo id, uploader username, date and time of upload, privacy setting, and a caption. Photo ID will generate an alphanumeric string to uniquely identify the photo in conjunction with the username of the uploader. This will serve as the key together with the foreign key from the User entity. Here are some differences that are not included in our current design, but are in the requirements. The specifications mention a dislike feature and a rating feature; however, Instagram has no such features and only offers the like capability. The user may take away a like given to a photo, but

may not rate or dislike a photo. Here there is also a numViews and numLikes that is also updated similarly as above in user for their numFollows.

Comment Entity: A comment shall be composed of the following attributes: the comment id, the comment itself, a photo id, author's username, and the date/time in which the comment was made. Photo Id and Username serve as foreign keys to properly identify comments along with its comment Id. Here we also have the numLikes attribute with similar purpose as above.

**Relationship:**

Photo Likes: Keeps track if a user likes a particular photo and enables us to keep track of the number of likes to a photo. A photo can have no likes or many likes. A user can like any number of photos or no photos.

Tagged: Keeps track of the users that are tagged in a photo. Here lies another difference between our implementation and the original specifications. The only form of tag support that is enabled on Instagram is user tags, not category tags that seem to be alluded to by the project specifications. A user can be tagged in many photos or no photos and a photo can tag as many users as it wishes or none.

Views: Keeps track of the users that have requested to view a photo. This helps us keep track of which photo is the most viewed photo. Users do not have to have viewed any photos and a photo can also have no views

Publishes: Keeps track of which user uploaded which photo and helps us keep track of how many photos a user has uploaded and who has the most uploaded. A photo must have a publisher, while a user does not have to have any photos uploaded at all.

Commented On/ Makes Comment: At this current Time we have these relationships as two separate relationships. Originally we thought it was a ternary relationship and may return

back to it. A comment must have only one author and it must also only have one photo that it is commented on. A photo may have as many comments or no comments on it. A user may have as many comments or no comments created.

Comment Likes: Takes a similar being to photo likes as a user can like as many comments or none and a comment may have zero likes or many likes.

### **SQL vs NoSQL:**

The above relations and entities are planned to be contained in the SQL environment(Postgres). While the photos themselves will be contained in the NoSQL environment(MongoDB. This is because the photo data is large in comparison to the text data. Look up on statistics, account information and the like need to be accurate and available, but the photo data can be delayed. This takes into account the growth of the user base and the amount of photos being uploaded into the databases.

### **Schema:**

```
DROP TABLE IF EXISTS Users;
DROP TABLE IF EXISTS Comments;
DROP TABLE IF EXISTS Photos;
DROP TABLE IF EXISTS Views;
DROP TABLE IF EXISTS PhotoLikes;
DROP TABLE IF EXISTS Tagged;
DROP TABLE IF EXISTS CommentLikes;
DROP TABLE IF EXISTS Follows;
```

-----

--Entities Tables:

```
CREATE TABLE Users
```

```
(
 username VARCHAR(64) NOT NULL,
 email VARCHAR(64),
```

```
 pass VARCHAR(64) NOT NULL,
 numFollows BIGINT NOT NULL,
 PRIMARY KEY (username)
);
CREATE TABLE Photos
(
 photo_id VARCHAR(64) NOT NULL,
 publisher VARCHAR(64) NOT NULL,
 dates DATE NOT NULL,
 privacy BIT NOT NULL,
 description VARCHAR(512),
 numLikes BIGINT NOT NULL,
 numViews BIGINT NOT NULL,
 PRIMARY KEY(photo_id),
 FOREIGN KEY(publisher) REFERENCES Users(username) ON DELETE CASCADE
);
CREATE TABLE Comments
(
 comment_id BIGINT NOT NULL,
 comments VARCHAR(1024),
 username VARCHAR(64) NOT NULL,
 photo_id VARCHAR(64) NOT NULL,
 dates DATE NOT NULL,
 numLikes BIGINT NOT NULL,
 PRIMARY KEY(comment_id),
 FOREIGN KEY(username) REFERENCES Users(username) ON DELETE CASCADE,
 FOREIGN KEY(photo_id) REFERENCES Photos(photo_id) ON DELETE CASCADE
);
```

---

--Relation Tables:

```
CREATE TABLE Views
(
 username VARCHAR(64) NOT NULL,
 photo_id VARCHAR(64) NOT NULL,
 PRIMARY KEY(username,photo_id),
 FOREIGN KEY(username) REFERENCES Users(username) ON DELETE CASCADE,
 FOREIGN KEY(photo_id) REFERENCES Photos(photo_id) ON DELETE CASCADE
);
```

```
CREATE TABLE PhotoLikes
```

```
(
 username VARCHAR(64) NOT NULL,
 photo_id VARCHAR(64) NOT NULL,
 PRIMARY KEY(username, photo_id),
 FOREIGN KEY(username) REFERENCES Users(username) ON DELETE CASCADE,
 FOREIGN KEY(photo_id) REFERENCES Photos(photo_id) ON DELETE CASCADE
);
```

--all the people who are tagged in the photo

```
CREATE TABLE Tagged
```

```
(
 username VARCHAR(64) NOT NULL,
 photo_id VARCHAR(64) NOT NULL,
 PRIMARY KEY(username, photo_id),
 FOREIGN KEY(username) REFERENCES Users(username) ON DELETE CASCADE,
 FOREIGN KEY(photo_id) REFERENCES Photos(photo_id) ON DELETE CASCADE
);
```

--User Likes a Comment

```
CREATE TABLE Likes
```

```
(
 username VARCHAR(64) NOT NULL,
 comment_id BIGINT NOT NULL,
 PRIMARY KEY(username, comment_id), FOREIGN KEY(username) REFERENCES
Users(username) ON DELETE CASCADE,
 FOREIGN KEY(comment_id) REFERENCES Comments(comment_id) ON DELETE
CASCADE
);
```

--1 follows 2

```
CREATE TABLE Follows
```

```
(
 username1 VARCHAR(64) NOT NULL,
 username2 VARCHAR(64) NOT NULL,
 PRIMARY KEY(username1, username2),
 FOREIGN KEY(username1) REFERENCES Users(username) ON DELETE
CASCADE,
 FOREIGN KEY(username2) REFERENCES Users(username) ON DELETE CASCADE
);
```

```
=====

```

-- INSERT DATA STATEMENTS --

-----

--Entity Data

COPY Users (

    username,  
    email,  
    pass,  
    numFollows

)

FROM 'Users.csv'

WITH DELIMITER ',' NULL AS "

CSV HEADER;

COPY Photos (

    photo\_id,  
    publisher,  
    dates,  
    privacy,  
    description,  
    numLikes,  
    numViews

)

FROM 'Photos.csv'

WITH DELIMITER ','

CSV HEADER;

COPY Comments(

    comment\_id,  
    comments,  
    username,  
    photo\_id,  
    dates,  
    numLikes

)

FROM 'Comments.csv'

WITH DELIMITER ',' NULL AS "

CSV HEADER;

---

--Relations Data

```
COPY Views (
 username,
 photo_id
)
FROM 'Views.csv'
WITH DELIMITER ','
CSV HEADER;
```

```
COPY PhotoLikes(
 username,
 photo_id
)
FROM 'PhotoLikes.csv'
WITH DELIMITER ','
CSV HEADER;
```

```
COPY Tagged(
 username,
 photo_id
)
FROM 'Tagged.csv'
WITH DELIMITER ','
CSV HEADER;
```

```
COPY Likes (
 username,
 comment_id
)
FROM 'Likes.csv'
WITH DELIMITER ','
CSV HEADER;
```

```
COPY Follows(
 username1,
 username2
)
FROM 'Follows.csv'
```



WITH DELIMITER ','

CSV HEADER;

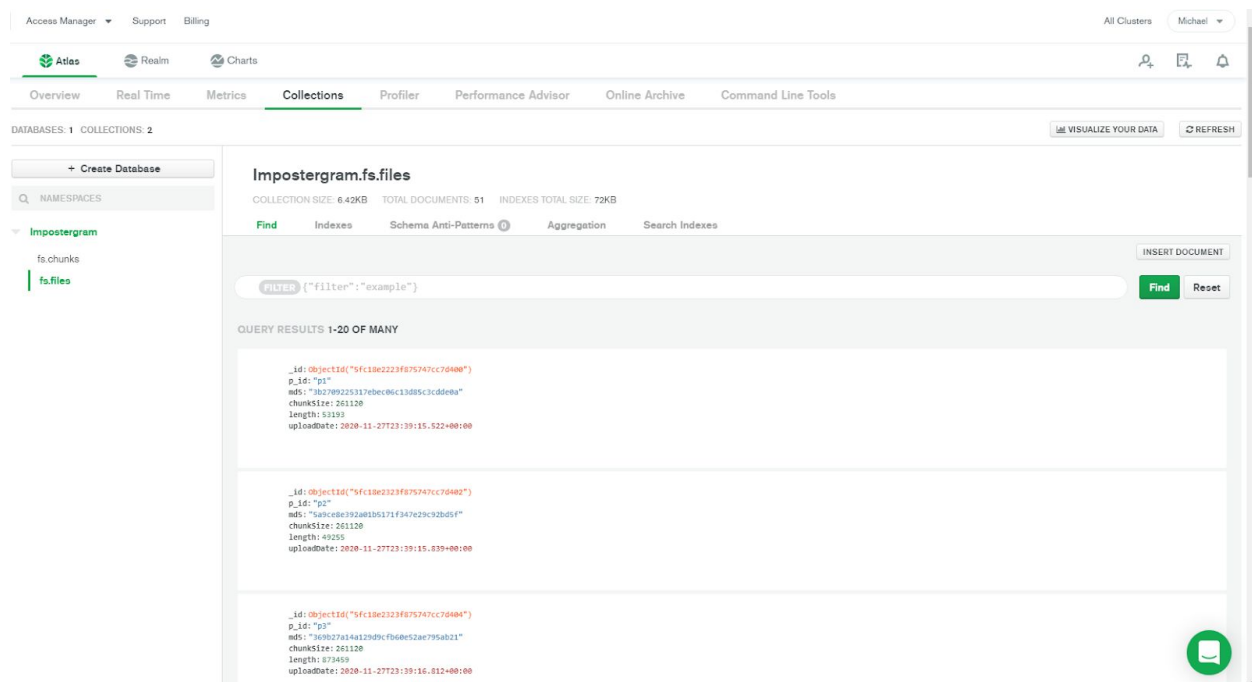
**Justifications:**

- Dropping tables if the table already exists.
- User: Username is the primary key. It cannot be null. Email can be null as it could not be verified and is pending. Passwords must exist.
- Comments: Id must exist and is a primary key along with the foreign keys of photoID and username. Comment can be a blank comment. A comment must have been created at some point in time. If the user that created the comment or if the photo that the comment existed on gets deleted then the comment is also deleted.
- Photos: Photo id is the primary key and must exist in conjunction with its uploader. All other data fields must exist except for a description.
- Relationship Tables: These exist to tie our entities together and allow us to make specific queries. These primary keys are made up by the foreign keys inherited from the related entities.
- If a photo is deleted from our database then any associated row containing that photo\_id will also be deleted.

**Implementation:**

**Postgresql:** We decide to create and host our own instance of a postgresql database to handle our user data and photo data. To accomplish this, we first created a virtual machine using virtualbox in order to get a linux environment using a Ubuntu distro. We then installed postgres and set up a local database. The reason for switching to a linux environment is because there were potential compatibility issues with working on windows, so in order to avoid issues with postgres or other future design choices we decided to utilize linux early on.

**MongoDB:** Instead of hosting a local instance of a postgres database we decided to utilize MongoDB's online cloud database hosting service Atlas. We created our database through this service and it is hosted on AWS servers. While we lose out on certain aspects that come with local hosting, we gain the reliability and features that come with hosting through AWS and MongoDB.



**Python:** We decided to create our project using python because we knew of its popularity and due to this popularity the likelihood of good documentation and APIs to interact with postgres and mongodb was high. We also took this opportunity to learn and become more familiar with python as we have worked with the language before, but at the time it was not our primary language.

**Psycopg2:** This is the python PostgreSQL database adapter that we chose to utilize to handle our SQL database calls.

```
import psycopg2

class post_db:
 def __init__(self):
 self.conn = None
 try:
 #connect to the postgresql db
 #print ("Attempting to create connection.")
 self.conn = psycopg2.connect(
 host="localhost",
 database="impostergram_db",
 user="team2",
 password="179g"
)
 #print ("Successfully made connection")

 except (Exception, psycopg2.DatabaseError) as error:
 print(error)
```

*Class object in order to establish the connection to our local database server when necessary.*

```
def close_connection(self):
 if self.cur is not None:
 self.cur.close()
 #print("Closing cursor")
 if self.post.conn is not None:
 self.post.conn.close()
 if self.post is not None:
 del self.post
 self.conn_closed = True
```

*Function to handle closing the cursor and connection to our local database.*

```

if validity:
 self.cur.execute("SELECT username FROM Tagged WHERE username = %s AND photo_id = %s", (username, self.__photo_id))
 if(self.cur.rowcount > 0):
 print("This user is already tagged.")
 loop = False
 continue
 else:
 self.cur.execute("INSERT INTO Tagged (username,photo_id) VALUES (%s, %s)", (username, self.__photo_id))
 self.post.conn.commit()

```

*Sample of some psycopg2 interaction with our postgres db found in our tagged.py.*

**Pymongo:** This is the python distribution that contains tools dedicated to working with MongoDB. We used this when we needed to interact with the photos themselves.

```

cluster = MongoClient("mongodb+srv://team2:179g@cluster0.fmq4y.mongodb.net/Impostergram?retryWrites=true&w=majority") #connects to our mongodb server
db = cluster["Impostergram"] #specifies the impostergram cluster
collection = db["fs.files"]
count = collection.count_documents({})

```

*Example of us using pymongo to establish a connection with our mongodb server. This is found in upload.py and we are creating a cursor to our fs.files collection specifically in order to acquire a count of how many photos are currently on our database.*

**Gridfs:** This is a subset found within pymongo and MongoDB in order to handle large data sizes. MongoDB normally has a size limit on what they will store, so in order to circumvent this limit, Gridfs breaks down the file into smaller binary file chunks and stores those along with a file that ties these chunks together. When the file is requested, altered, or deleted the associated chunks are also affected. This file system is handled automatically by MongoDB.

```

self.cur.execute("INSERT INTO Photo
self.post.conn.commit()
fs = gridfs.GridFS(db)
f = open(pathFile, 'rb')
img_id=fs.put(f,p_id=pid)
print("Upload Successful.")

```

*Here is an example of our gridfs calls in order to upload a photo after we have used python native open function to read the image as a binary file. Found in upload.py*

**Sample Data:**

For our sample data we created some “lore” related stuff with our group of friends; however, the specifics do not pertain directly to the project or its scope therefore we will skip the explanation of why we generated the data the way we did. We did keep some notes on some of the interconnected nature of our data and some statistics for testing purposes.

Here are the notes:

Belinda- Person with the photo that has the most likes, and views ,and comments (21 comments)

-Photo id 1 and has 30 likes

-No comments

-28 follows

Frank- has no photos uploaded

Kyle- has the most comments posted (15 comments)

Simping for Belinda

Mario- has the comment with the most likes (28 Likes)

After Kyle's last comment on p1

Bob- has no comments

- has the most followers

Needs to have one blank general comment.

Kyle leaves a blank comment to Calvin's photo

Jessie- responds to tohsaka's photo

Mustafa- comments on mychul's photo

Belinda has 28 Follows

Bob has 29 Follows

P1 - 30 Likes

P5 - 4 Like

P24 - 8 Likes

P18 - 5 Likes

P28- 2

P1- 21 comments

P5 - 4 comments

P20 - 1 comment

P24 - 3 comments

P25 - 1 comment

*End of Notes*-----

## **User Manual:**

### Login Step

- You can login to your account with username and password.
- If you don't have your own account, you need to sign up.
  - To sign up, you need to use a username which does not collide with other username.
  - You need to put a password and email address.

### Main Menu:

- You navigate to the Photo Menu to search for photos.
- You can search other users in the User Search.
- You can follow/unfollow Users.
- You can view top posts and your feed.
  - There are some options of top posts such as "Top 5 Users", "Top 5 Photos", and "Top 5 Photos by likes"
- You can also upload photos from your computer or delete the photo which you uploaded before.

### Photo Menu:

- There are many options to search photos such as "tagged username", "# of likes", "date", "uploader's username", and "photo id".
- After searching, you can see the photo which you search with an option.
- After viewing the photo, you can do some functions to the searched photo.
  - There are 4 kinds of functions you can use such as "Like/Unlike the photo", "Tag/Untag a user", "View/Make a comment", and "Download the photo"

- If you choose the “View/Make a comment” option, you can navigate to the Comments Menu.

#### Comments Menu:

- You can enter Comments Menu by Photo Menu(“View/Make a comment”)
- Make Comments.
- View Comments.
- After viewing the comment, you can do the “Like/Unlike a comment” option.

#### User Search

- You can use variable options to search for specific users such as “Username”, “Description”, “# of follow”, and “# of photos published”.
- After you finish the user search, you can go back to the Main Menu and follow/unfollow other users who you search for.

#### Upload

- To upload the photo from your computer, you need to set up your photo in the “Uploads” folder.
- Currently only .jpg are configured to be supported by our program.

#### Download

- To download the photo that you search for on your computer it will be saved to your “Downloads” folder.
- Downloaded photos will be saved as .jpg photo format.

#### Delete

- To delete a photo you must be the owner and specify the correct photo id.
- If you delete a photo, you can’t recover that photo.

